

Module Seven

Architecture and Design

This module describes the architectural features that support the security aspects of a trusted system. Some of these features must be present in the architecture of a system in order for it to enforce certain aspects of the system's security policy. The architecture requirements from the TCSEC are given and some actual architectural features are described.

Module Learning Objectives

The material presented in this module describes architectures that support the mechanisms described in Module 6. Upon completion of this module, the student should:

1. Be familiar with various architectural features that can be used to support the enforcement of a trusted system's security policy.
2. Understand the TCSEC architecture requirements.
3. Understand how multistate machines work and how they support the system's security policy.
4. Understand the security aspects of file system and memory management.
5. Understand how the TCB and its hardware devices are protected by the architecture.
6. Be familiar with the notions of modularity, data hiding, abstraction, layering and least privilege. Be aware of how they affect the ability to examine the TCB and verify that it correctly enforces the security policy.

Overview

With regards to architecture, the TCSEC requires mechanisms to support the enforcement of the security policy by the TCB, as well as the use of design techniques that enhance the understandability of the TCB and increase the level of assurance that the TCB operates correctly. The architecture requirements flow from the reference monitor requirements for the TCB to be 1) tamperproof, so that non-TCB subjects cannot interfere with the correct operation of the TCB, 2) impossible to bypass, so that non-TCB subjects cannot simply go around the TCB and violate the security policy, and 3) verifiable, so that developers and evaluators can examine the TCB and determine that it correctly enforces the security policy.

TCBs at B2 or above are required to have a well-defined user interface. The interface must be completely described so that a top-level specification (descriptive (DTLS) and/or formal (FTLS)) can be written and shown to be an accurate representation of the interface. Verification arguments about how the TCB implements the reference monitor concept and why the TCB is complete, tamper-proof, and verifiable are then made with respect to the DTLS/FTLS.

Memory Protection, I/O Access Control, and Asynchronous Event Handling

The architecture of the TCB should provide support for memory protection. This is necessary to protect processes from each other, as well as to protect the

Module Seven

system integrity by isolating the TCB code and data objects(preventing interference or tampering with the TCB code and data by non-TCB subjects). There should also be support for ensuring that the TCB can mediate access to I/O devices. This is necessary to ensure that the TCB can enforce the security policy with respect to I/O devices. There must also be support for the secure handling of asynchronous events such as interrupts and traps.

Execution Domains

An approach by which the TCB protects its code and data and also ensures that it controls the I/O devices is to use execution domains. The TCB must maintain a domain for its own execution that protects it from external interference or tampering. In a two-state machine, a process may run in either of two domains, privileged or unprivileged. A process running in the privileged mode will be allowed by the hardware to use privileged instructions (e.g., I/O instructions) and to gain access to privileged sections of memory (e.g., TCB code and data) that are denied to processes running in non-privileged mode. Some systems, not having privileged I/O instructions, protect devices by only allowing privileged processes to access the address space containing the devices.

The two-state machine is an example of hierarchical execution domains, where privileged subjects get access to everything available to unprivileged subjects as well as resources denied to unprivileged subjects. An architecture where there are more than two hierarchical execution domains is called a ring architecture because it is commonly pictured as a set of concentric rings with the most privileged ring in the center. Each ring has access to its own resources and to the resources available to the rings outside it but no access to the resources of the more privileged rings inside it.

Process Management

The architecture of a TCB must also provide mechanisms for process management. TCBs at B1 or above must have the ability to isolate processes from each other. This is necessary to be able to enforce the security policy with respect to communication between subjects. The TCBs at B1 or above must have some control over process scheduling because a subject that knows about the state of processes at all different levels, and is able to modify their state, must be trusted. TCBs at B1 or above must be able to control a process's address space (i.e., to which objects the process has access). Interpretation C1-CI-04-85 [INTERP94] explains that although most examples of process isolation have relied upon a hardware-based architecture, a properly implemented software architecture would satisfy the B1 requirement.

Typically, the address space of a process is limited to no more than a few hundred objects at a time. Therefore, the process can use a small set of reusable names to identify the objects in its address space. For example, a process must know the large system-wide names of the files to which it wants to request access, but when the access is granted (adding a file to the process's address space), the file is assigned a file descriptor which is used to subsequently identify the file when reading or writing it.

Module Seven

In some systems, where code and/or data is shared between processes, an issue arises in resolving references to shared objects. An example of this might be the need to make calls to a shared run-time library located in a separate segment. In some systems, these references are resolved at compile-time. In other systems, the references are resolved at run-time, which is called binding. Some systems resolve all the references by invoking the binder or linker when the process is loaded, but before control is transferred to its main procedure. Other systems (e.g., Multics) have mechanisms for invoking the binder to resolve each unresolved reference as it is actually needed. This last approach is called dynamic binding.

Memory Management

Some types of hardware support for memory management are commonly used to support the TCB's enforcement of the security policy. In segmented (or paged) systems, the TCB can control which segments (or pages) to which a process is granted access, and can trust the hardware to limit the process to only those segments (or pages) of memory. Thus, the lower (hardware) layer enforces the access decision after mediation, which avoids having the TCB software mediate every memory reference made by each untrusted process. TCBs at B2 or above are required to use hardware features such as segmentation or paging to support separate objects with hardware-enforced modes of access (viz., read, write, execute, etc.).

Direct memory access I/O introduces some complexities to the enforcement of the security policy. For example, if a process makes an I/O request to read into a segment, and the segment is removed from the process's address space prior to the completion of the I/O operation, then the device might write to an area of memory outside of the address space of the process on whose behalf it's working. TCBs at C2 or above must prevent this situation, because they are required to isolate all resources being protected. TCBs at B1 or above are further required to control the address spaces of all processes.

Another area of memory management that must be controlled by the TCB is the use of caches. Of particular interest are caching schemes used by multilevel processes that communicate with untrusted processes with different security clearances. The multilevel process must be examined to show that the untrusted processes cannot obtain information about each other based on the contents of the cache. Special analysis would be required to assess the security implications of uncommon caching schemes such as associative memory, where the items in the cache are addressed by content rather than location.

If the TCB executes in a multiprocessor environment, the architecture must provide mechanisms for objects to be shared between the processors. If there is shared memory, then there must be a means for the processors to keep consistent descriptor tables, or maps of the shared memory. A common mechanism to support this is to use memory locking instructions that read and write in one atomic hardware operation.

Module Seven

Modularity, Layering, Data Abstraction, and Complexity Minimization

The structure of the TCB and the design techniques used to develop it play an important part in determining the level of assurance that the TCB correctly enforces the security policy. TCBs at B2 or above must be divided into well-defined modules (with data hiding being a required criterion of module development for systems at B3 or above). It is important to remember that *modularity* requirements apply to the entire TCB. In particular, the trusted processes and the libraries that are linked into the trusted processes must also meet the same standards as the security kernel. Although a requirement for complexity minimization is not imposed until B3, there has been a decision made that the TCB for B2 systems may not contain extraneous (dead) code. This has been further interpreted to allow some "unused" code; such as a service provided by a type manager which turns out to not need to be exercised by the actual TCB implementation. Another decision that has not yet been put in the form of an official Interpretation is the decision that the modularity standards must also be met by any unused code that is linked into the TCB.

The concept of *layering* is required at B3 or above, where 1) layers of the TCB know about the interfaces and depend on the services of layers below, but know nothing about and do not depend on the correct functioning of the layers above, 2) each layer of the TCB is protected from tampering by the layers above, and 3) layers cannot violate the portions of the security policy enforced by the layers below, is also a required design technique in the development of TCBs at B3 or above. Layering not only facilitates the verification of the correctness of the TCB by allowing examination of one layer at a time, but it also simplifies the RAMP by allowing the higher layers to be modified (or perhaps even "chopped off" and replaced) for new releases without the need to redesign (or reverify) the lower layers.

The concept of *data abstraction* is also required in the design of TCBs at B3 or above. For example, the design might make use of a stack object, with the operations PUSH and POP, so that the use of the stack is easier to understand than if the design described an array of words and a pointer and the algorithms used to temporarily store words in the array.

TCBs at B3 or above must also enforce the concept of *complexity minimization* in their design. Significant system engineering must be directed toward minimizing the complexity of the TCB and excluding modules from the TCB that are not protection-critical. This effort is necessary to enable the evaluators to more thoroughly examine and understand the TCB.

Although layering, abstraction and data hiding are not required until B3, they are good design techniques for any large system development. Because these architectural features aid in the understandability and maintainability of the system, their use would surely facilitate the evaluation of a TCB at any class, and simplify its subsequent RAMP effort. Complexity minimization would also facilitate the evaluation of a system at any class, but is seldom done for systems requiring less than a B3 rating because the process of excluding non-protection-critical functionality from the TCB has the effect of significantly increasing the complexity of the system outside of the TCB.

Module Seven

Least Privilege

The least privilege principle (LPP) requires that every subject must operate with the minimum set of privileges necessary to accomplish its task. TCBs at B2 or above must enforce the LPP. The LPP is both a design technique for the architecture of the TCB, and a rule to be enforced by the TCB on non-TCB processes. As a design criterion for the TCB, it requires that the TCB not assign tasks to TCB subjects with more privilege than necessary to accomplish the task (e.g., not assign a task to a multilevel process that does not require reading and writing of objects with a range of sensitivities). An example of a LPP rule enforced by some systems on their users is that a user can be an active member of only one group at a time, and must switch out of the current group and into a second group to access files only accessible to members of the second group (as opposed to allowing the user to be an active member of both groups at the same time).

The LPP has been one of the most frequently misunderstood requirements for B2 and above systems. The two most common associations made with the LPP are the separation of administrative roles and the decomposition of the UNIX "superuser" role. Unfortunately, those two aspects do not completely cover the full scope of the LPP requirement. The LPP is intended to limit the damage that can result from accident, error, or unauthorized use. It is imposed as a requirement on individual subjects -- not users or programs. One of the objectives of adhering to the LPP is that any individual responsible for implementing, reviewing, or maintaining the TCB code must be able to determine the privileges that will be in effect at each point in the code and the potential impact of making any alterations to the code or set of privileges in effect at that point in the code.

Although there have not been any announced Interpretations of the LPP, there have been some decisions made for specific evaluations stating that implementations which associate privileges with specific trusted programs being executed on behalf of specific trusted users would be unlikely to meet the full LPP requirement unless the systems have the capability to further restrict the scope of a privilege. The technique used for implementing this fine a granularity of control over privilege is often referred to as "*privilege bracketing*."

The intent of privilege bracketing is to limit the scope of the potential effect of a privilege to as small a code segment as possible. The most common implementation of privilege bracketing is to identify each of the specific instances of system calls that will require the use of privileges, enable the appropriate privileges immediately before making the identified system call, and then disable the privilege immediately upon returning from the system call. As a result, most of the code for a trusted subject executes without any privileges enabled. There have been exceptions made to bracketing at the system call level to allow for the use of public libraries and certain highly specialized privileges.

Module Seven

Relevant Trusted Product Evaluation Questionnaire Questions

2.3 HARDWARE ARCHITECTURE

If this evaluation is for a family of hardware, the following questions should be answered for each member of the hardware family. You may choose to answer each question for each member of the family, or answer each question for a baseline family member and point out the difference for each of the remaining family members.

C1:

1. Provide a high-level block diagram of the system. The diagram should at least depict various Central Processor Units (CPUs), memory controllers, memory, I/O processors, I/O controllers, I/O devices (e.g., printers, displays, disks, tapes, communications lines) and relationship (both control flow and data flow) among them.
2. (a) Describe the portions of the system (if any) which contain microcode. (b) How is this microcode protected and loaded?
3. (a) Provide a list of privileged instructions for your hardware. (b) Provide a brief description of each privileged instruction.
4. For each privileged instruction, provide the privileges required to execute the instruction. (Examples of privileges include the machine state, the executing ring/segment/domain/privilege level, physical memory location of the instruction, etc.)
5. How does the process address translation (logical/virtual to physical) work in your system?
6. (a) How does I/O address translation work for the Direct Memory Access (DMA) controllers/devices? (b) Identify if the address translation is done through the memory address translation unit or if the logic is part of the controller. (c) How are the address translation maps and/or tables initialized?
7. Describe the hardware protection mechanisms provided by the system.
8. Describe what hardware mechanisms are used to isolate the TB from untrusted applications.
9. (a) What are the machine/processor states supported by the system? (b) How are the states changed? (c) What data structures are saved as part of the processor state?
10. List all the (a) interrupts and (b) traps (hardware and software). (c) How are they serviced by the system?

B1:

11. Provide a high-level block diagram of a CPU. The diagram should explain the relationship among the elements such as: Instruction Processor, Microsequencer, Microengine, Memory, Cache,

Module Seven

Memory Mapping or Address Translation Unit, I/O devices and interfaces.

12. Describe the hardware isolation mechanisms for the process memory (e.g., rings, segments, privilege levels).
13. (a) Provide a description of the process address space. (b) When and (c) how is it formed? (d) How does the software use this mechanism, if it does at all?

2.4 SOFTWARE

The TCB software consists of the elements that are involved in enforcing the system security policy. Examples of *TCB elements* include: kernel, interrupt handlers, process manager, I/O handlers, I/O manager, user/process interface, hardware and command languages/interfaces (for system generation, operator, administrator, users, etc.). The security kernel is the hardware, firmware and software elements of the TCB that are involved in implementing the reference monitor concept, i.e., the ones that mediate all access to objects by subjects.

C1:

3. Describe the hardware ring/domain/privilege level/memory segment / physical location where each TCB element resides.
4. Describe the hardware ring/domain/privilege level/memory segment / physical location where the user processes reside.
8. (a) List the process states and (b) briefly state conditions under which transition from one state to another occurs.
9. Briefly describe process scheduling.
10. Describe all interprocess communications mechanisms.
11. (a) Describe the file management system. This should include the directory hierarchy, if any, directory and file attributes. (b) Also identify all system directories and files, and (c) their access attributes.
12. How are (a) I/O devices and (b) their queues (if any) managed?
13. How are the (a) batch jobs and (b) their queues managed?
14. What software engineering tools and techniques were used for the TCB design and implementation?

C2:

15. Describe the interfaces (control and data flow) among the TCB elements.
16. Describe the interfaces between the kernel and the rest of the TCB elements.
17. Describe how the process states are manipulated by the TCB.
18. (a) Describe the data structures for a process context. Describe both (b) hardware and (c) software mechanisms used to manipulate/switch the process context.

Module Seven

B1:

19. (a) List software mechanisms that are used to isolate and protect user processes. (b) Provide a brief description of each mechanism.
20. (a) Describe various elements of the process address space and (b) their location in terms of ring/domain/privilege level/segment/physical memory.

B2:

22. How was the modularity requirement achieved and implemented?
24. (a) Is the TCB layered? (b) If yes, how many layers are in the TCB? Provide a brief description of (c) modules and (d) functions in each layer. (e) How are the lower layers protected from higher layers?

B3:

25. How does the architecture limit or restrict the ability of untrusted code to exploit covert channels?
26. How is the least privilege required achieved and implemented?
28. How was the data abstraction and information hiding requirement achieved and implemented?

2.13 OTHER ASSURANCES

C1:

7. (a) Does the system have a degraded mode of operation? (b) What can cause this to occur? (c) How long can the system keep running in this mode? (d) How does an operator get the system back to full operation? (e) What security related services are provided in the degraded mode? (f) What security related services are not provided?

B3:

14. (a) How does the system recovery work? What system resources (e.g., memory, disks blocks, files) are protected (b) prior to and (c) during the system recovery? (d) How are they protected? (e) What resources are not protected?

Required Readings

TCSEC85 National Computer Security Center, *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

Sections 2.1.3.1.1, 2.2.3.1.1, 3.1.3.1.1, 3.2.3.1.1, 3.3.3.1.1 and 4.1.3.1.1 describe the system architecture requirements, which are summarized on page 105.

INTERP94 National Computer Security Center, *The Interpreted TCSEC Requirements*, (quarterly).

Module Seven

The following Interpretation is relevant to system architecture:

C1-CI-04-85 System Architecture

- Gasser88 Gasser, M., *Building a Secure Computer System*, Van Nostrand Reinhold Co., N.Y., 1988.
- Chapter 4 gives an introduction to system structures, including reference monitors. Chapter 8 discusses hardware security mechanisms such as process support, memory protection, execution domains, I/O access control, and multiprocessor support. Chapter 10 discusses the security kernel architecture and implementation strategies. Chapter 11 discusses operating system issues such as layering, protected subsystems, and secure file systems. All four should be read in their entirety. Section 5.4 should be read for an introduction to the concept of least privilege.
- Arnold92 Arnold, J.L. et. al., "Assessing Modularity in Trusted Computing Bases," *Proceedings of 15th National Computer Security Conference*, Vol 1, pp. 44-56, October 1992
- This paper summarizes the findings of a working group that was established to define and clarify modularity criterion contained within the System Architecture requirement for Class B2 of the TCSEC.
- Intel83 Intel Corporation, *Intel iAPX 286 Operating Systems Writer's Guide*, 1983.
- Chapter 1, pp. 1-1 to 1-8 and pp. 2-1 to 2-20, provides an introduction to the notions of non-circumventability and isolation of a reference monitor built on the iAPX 286 architecture. Topics covered include: asynchronous event handling, memory protection, execution domains, ring architectures, and memory management.
- Parnas72 Parnas, D.L., "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, Vol. 15, No. 12, pp. 1053-1058, December 1972.
- This paper should be read in its entirety. It illustrates the notions of modularity and data hiding necessary to design software (e.g., TCB software) that is understandable.
- Saltzer78 Saltzer, J., "Naming and Binding of Objects," in *Operating Systems -- An Advanced Course*, lecture notes in Computer Science, Vol. 105, Appendix A of Chapter 3.A, Springer Verlag, 1978.
- This appendix illustrates the notions of naming and binding through a case study of the Multics architecture of shared-object addressing. The chapter this appendix came from is listed in the other readings for this module.

Module Seven

Supplemental Readings

- Hecht87 Hecht, M., et. al., "UNIX without the Superuser," *USENIX Conference Proceedings*, pp. 243-256, June 1987.

This paper presents two distinct, but related ideas. First, it shows how the system (i.e., superuser) privilege can, and should, be partitioned into specific privileges for each TCB action to help enforce least privilege. Second, it shows how the roles of system administration can be partitioned in a power hierarchy that balances the security administrator role against that of the auditor.

- Knowles87 Knowles, F. and Bunch, S., "A Least Privilege Mechanism for UNIX," *Proceedings of the 10th National Computer Security Conference*, pp. 257-262, September 1987.

This paper describes a privilege control mechanism for the UNIX OS. The mechanism is intended to satisfy the B2 requirement of least privilege, and to provide fine-grained control over access by users to services and objects. The mechanism is largely independent of other security-related features and is useful as an incremental addition to a less secure UNIX.

- Schroed77 Schroeder, M.D., et al, "Multics Kernel Design Project," *Proceedings of the Sixth ACM Symposium on Operating System Principles*, November 1977.

This paper provides a good discussion of layering issues as encountered in the kernelization of Multics.

Other Readings

- Bondi89 Bondi, J.O. and Branstad, M.A., "Architectural Support of Fine-Grained Secure Computing," *Proceedings of the 5th IEEE Computer Security Applications Conference*, pp. 121-130, December 1989.

This article outlines an approach to incorporating a large portion of the security policy enforcement into the architecture of the hardware component of the TCB.

- Smith86 Smith, T.A., "User Definable Domains as a Mechanism for Implementing the Least Privilege Principle," *Proceedings of 9th National Computer Security Conference*, pp. 143-148, September 1986.

User definable domains, as developed in this paper, allow the principle of least privilege to be implemented completely, thus providing users with significantly greater protection against the threat of Trojan Horses and viruses.